● **Nosql Databases**

NoSQL is a class of database management systems (DBMS) that do not follow all of the rules of a relational DBMS and cannot use traditional SQL to query data. The term is somewhat misleading when interpreted as "No SQL," and most translate it as "Not Only SQL," as this type of database is not generally a replacement but, rather, a complementary addition to RDBMSs and SQL.

**● Types of Nosql Databases**

1. **Key-value model**—the least complex NoSQL option, which stores data in a schema-less way that consists of indexed keys and values. *Examples: Cassandra, Azure, LevelDB, and Riak.*
2. **Column store**—or, wide-column store, which stores data tables as columns rather than rows. It’s more than just an inverted table—sectioning out columns allows for excellent scalability and high performance. *Examples: HBase, BigTable, HyperTable.*
3. **Document database**—taking the key-value concept and adding more complexity, each document in this type of database has its own data, and its own unique key, which is used to retrieve it. It’s a great option for storing, retrieving and managing data that’s document-oriented but still somewhat structured. *Examples: MongoDB, CouchDB.*
4. **Graph database**—have data that’s interconnected and best represented as a graph? This method is capable of lots of complexity. *Examples: Polyglot, Neo4J.*

● **CAP Theorem**

The CAP theorem is a tool used to makes system designers aware of the trade-offs while designing networked shared-data systems. CAP has influenced the design of many distributed data systems. 

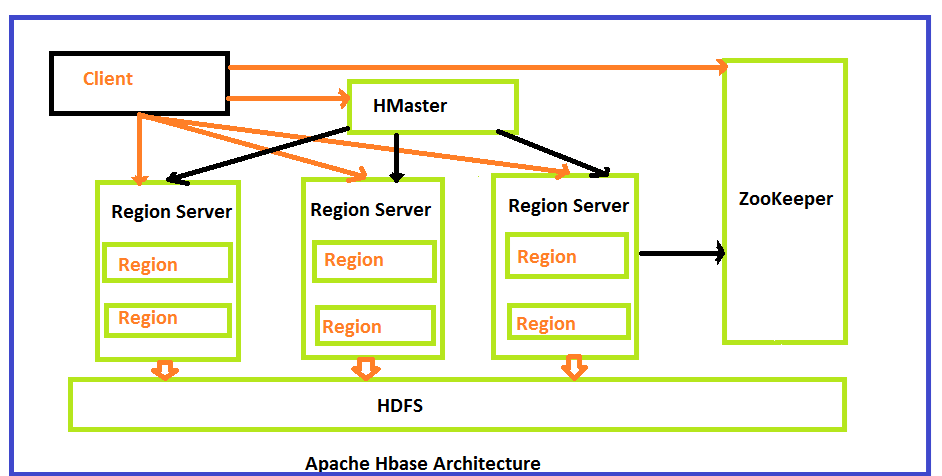
The theorem states that networked shared-data systems can only guarantee/strongly support two of the following three properties:

* Consistency - A guarantee that every node in a distributed cluster returns the same, most recent, successful write. Consistency refers to every client having the same view of the data. There are various types of consistency models. Consistency in CAP (used to prove the theorem) refers to linearizability or sequential consistency, a very strong form of consistency.
* Availability - Every non-failing node returns a response for all read and write requests in a reasonable amount of time. The key word here is every. To be available, every node on (either side of a network partition) must be able to respond in a reasonable amount of time.
* Partition Tolerant - The system continues to function and upholds its consistency guarantees in spite of network partitions. Network partitions are a fact of life. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

The CAP theorem categorizes systems into three categories:

* CP (Consistent and Partition Tolerant) - At first glance, the CP category is confusing, i.e., a system that is consistent and partition tolerant but never available. CP is referring to a category of systems where availability is sacrificed only in the case of a network partition.
* CA (Consistent and Available) - CA systems are consistent and available systems in the absence of any network partition. Often a single node's DB servers are categorized as CA systems. Single node DB servers do not need to deal with partition tolerance and are thus considered CA systems. The only hole in this theory is that single node DB systems are not a network of shared data systems and thus do not fall under the preview of CAP.
* AP (Available and Partition Tolerant) - These are systems that are available and partition tolerant but cannot guarantee consistency.

● HBase Architecture



HBase architecture consists mainly of four components

* HMaster
* HRegionserver
* HRegions
* Zookeeper

**HMaster:**

HMaster is the implementation of Master server in HBase architecture. It acts like monitoring agent to monitor all Region Server instances present in the cluster and acts as an interface for all the metadata changes. In a distributed cluster environment, Master runs on NameNode. Master runs several background threads.

The following are important roles performed by HMaster in HBase.

* Plays a vital role in terms of performance and maintaining nodes in the cluster.
* HMaster provides admin performance and distributes services to different region servers.
* HMaster assigns regions to region servers.
* HMaster has the features like controlling load balancing and failover to handle the load over nodes present in the cluster.
* When a client wants to change any schema and to change any Metadata operations, HMaster takes responsibility for these operations.

Some of the methods exposed by HMaster Interface are primarily Metadata oriented methods.

* Table ( createTable, removeTable, enable, disable)
* ColumnFamily (add Column, modify Column)
* Region (move, assign)

The client communicates in a bi-directional way with both HMaster and ZooKeeper. For read and write operations, it directly contacts with HRegion servers. HMaster assigns regions to region servers and in turn check the health status of region servers.

In entire architecture, we have multiple region servers. Hlog present in region servers which are going to store all the log files.

**HRegions Servers:**

When Region Server receives writes and read requests from the client, it assigns the request to a specific region, where actual column family resides. However, the client can directly contact with HRegion servers, there is no need of HMaster mandatory permission to the client regarding communication with HRegion servers. The client requires HMaster help when operations related to metadata and schema changes are required.

HRegionServer is the Region Server implementation. It is responsible for serving and managing regions or data that is present in distributed cluster. The region servers run on Data Nodes present in the Hadoop cluster.

HMaster can get into contact with multiple HRegion servers and performs the following functions.

* Hosting and managing regions
* Splitting regions automatically
* Handling read and writes requests
* Communicating with the client directly

**HRegions:**

HRegions are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families. It contains multiple stores, one for each column family. It consists of mainly two components, which are Memstore and Hfile.

**ZooKeeper:**

In Hbase, Zookeeper is a centralized monitoring server which maintains configuration information and provides distributed synchronization. Distributed synchronization is to access the distributed applications running across the cluster with the responsibility of providing coordination services between nodes. If the client wants to communicate with regions, the servers client has to approach ZooKeeper first.

● **HBase vs RDBMS**

* SQL databases are primarily called as Relational Databases (RDBMS); whereas NoSQL database are primarily called as non-relational or distributed database.
* SQL databases are table based databases whereas NoSQL databases are document based, key-value pairs, graph databases or wide-column stores. This means that SQL databases represent data in form of tables which consists of n number of rows of data whereas NoSQL databases are the collection of key-value pair, documents, graph databases or wide-column stores which do not have standard schema definitions which it needs to adhered to.
* SQL databases have predefined schema whereas NoSQL databases have dynamic schema for unstructured data.
* SQL databases are vertically scalable whereas the NoSQL databases are horizontally scalable. SQL databases are scaled by increasing the horse-power of the hardware. NoSQL databases are scaled by increasing the databases servers in the pool of resources to reduce the load.
* SQL databases uses SQL ( structured query language ) for defining and manipulating the data, which is very powerful. In NoSQL database, queries are focused on collection of documents. Sometimes it is also called as UnQL (Unstructured Query Language). The syntax of using UnQL varies from database to database.
* SQL database examples: MySql, Oracle, Sqlite, Postgres and MS-SQL. NoSQL database examples: MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j and CouchDb
* For complex queries: SQL databases are good fit for the complex query intensive environment whereas NoSQL databases are not good fit for complex queries. On a high-level, NoSQL don’t have standard interfaces to perform complex queries, and the queries themselves in NoSQL are not as powerful as SQL query language.
* For the type of data to be stored: SQL databases are not best fit for hierarchical data storage. But, NoSQL database fits better for the hierarchical data storage as it follows the key-value pair way of storing data similar to JSON data. NoSQL database are highly preferred for large data set (i.e for big data). Hbase is an example for this purpose.
* For scalability: In most typical situations, SQL databases are vertically scalable. You can manage increasing load by increasing the CPU, RAM, SSD, etc, on a single server. On the other hand, NoSQL databases are horizontally scalable. You can just add few more servers easily in your NoSQL database infrastructure to handle the large traffic.
* For high transactional based application: SQL databases are best fit for heavy duty transactional type applications, as it is more stable and promises the atomicity as well as integrity of the data. While you can use NoSQL for transactions purpose, it is still not comparable and sable enough in high load and for complex transactional applications.
* For support: Excellent support are available for all SQL database from their vendors. There are also lot of independent consultations who can help you with SQL database for a very large scale deployments. For some NoSQL database you still have to rely on community support, and only limited outside experts are available for you to setup and deploy your large scale NoSQL deployments.
* For properties: SQL databases emphasizes on ACID properties ( Atomicity, Consistency, Isolation and Durability) whereas the NoSQL database follows the Brewers CAP theorem ( Consistency, Availability and Partition tolerance )
* For DB types: On a high-level, we can classify SQL databases as either open-source or close-sourced from commercial vendors. NoSQL databases can be classified on the basis of way of storing data as graph databases, key-value store databases, document store databases, column store database and XML databases.